Constants, Traits, Static method, Static property, Namespaces in object oriented Programming.

# Class Constants:

Class constants provide the mechanism to hold fixed value in a program. Constant once defined can’t change in the program. Constants are helpful when we want values to be fixed during the program. Constant is not defined with ($) dollar sign. Constant are defined with the keyword **const** inside the class. Constants are case sensitive, however it is recommended to write the constant name in upper case letter. Let’s define a constant in a class.

**Code:**

<?php

class Welcome {

 const WELCOME\_MESSAGE = "Welcome to our website";

}

echo Welcome::WELCOME\_MESSAGE;

?>

**Output:**

Welcome to our website

We can also access constants without creating objects outside the class by using class name followed by the scope resolution operator **(::)** and then the constant name.

Here is the screenshot of the above code.



Figure

We can also access the constant inside the class by using **self** keyword along with scope resolution operator **(::) with** constant name.

**Code:**

<?php

class Welcome {

 const WELCOME\_MESSAGE = "Welcome to our website";

 public function welcome\_message(){

 echo self::WELCOME\_MESSAGE;

 }

}

$message=new Welcome;

$message->welcome\_message();

?>

**Output:**

Welcome to our website

We can access the constants inside and outside the class.

Here is the snippet of the above code.



Figure

# Static Methods:

If you want to access the methods in the context of class rather than the instance of class then static methods are used. Static method allows you to access the method on direct call without the creating of object.

To add a static method to your class use the static keyword as follows:

**Syntax:**

public static function (){

}

Let’s use the static method in a class.

**Code:**

<?php

class Message {

 public static function staticMethod() {

 echo "This is a static method";

 }

}

echo Message::staticMethod();

?>

**Output:**

This is a static method

In the above code we declare a class with name (Message).A functions with name (**staticMethod**) using the public access modifier along with the **static** keyword is used in the (Message) class. In the function we are printing a message (**This is a static method**).Outside the class we can access the static method by using class name followed by the scope resolution operator **(::) with** constant name.

Here is the illustration of above piece of code in Figure 3.



Figure

# Static Properties:

If you want to access the properties in the context of class rather than the instance of class then static properties are used. Static property allows you to access the property on direct call without the creating of object.

To add a static property to your class use the static keyword as follows:

**Syntax:**

public static $name;

Let’s use the static method in a class.

**Code:**

<?php

class Message {

 public static $property= "Im a property";

}

echo Message::$property;

?>

**Output:**

Im a property

In the above code we declare a class with name (Message).A property with name ($property) using the public access modifier along with the **static** keyword is assigned with a value (**Im a property**) .Outside the class we can access the static property by using class name followed by the scope resolution operator **(::) with** ($ )dollar sign and the property name.

Illustration can be seen in Figure 4.



Figure

# Traits:

PHP supports single inheritance that each child can be inherited by only one parent class. But if we want to use multiple classes in a class then traits are used. Traits have the mechanism to inherit multiple behaviors to a class. Traits are used to declare methods that can be used in multiple classes. A trait can have abstract and non abstract methods. Traits methods can use any access modifier.

**Syntax:**

trait traitName{

}

Traits are declared by using **trait** keyword.

**Code:**

<?php

trait hello{

 function sayhello(){

 echo "Hello";

 }

}

trait world{

 function sayworld(){

 echo "World";

 }

}

class Message{

 use hello,world;

}

$object =new Message;

echo $object->sayhello() . " " . $object->sayworld();

?>

In the above code we have defined two traits with name (**hello and world** and define two functions with name (**sayhello** and **sayworld**) which are printing the **Hello world .**We make a class by name (**Message** ) in which we use both the traits by using **use** keyword and separate the traits by using ( ,) comma. Then we make an instance of class by name ($object) and use the trait methods in the class. So in this way we can use multiple inheritances in PHP by using traits.

Here is the snapshot in Figure 5.



Figure

# Namespace:

Namespace allow you to use multiple classes with same name. Namespace can prevent collision. Namespace makes our code more organized. In PHP we can’t use the same name for the class as it says can’t declare the class again. We can differentiate the class name by giving prefixes but it’s very hard to manage these prefixes. So namespace are used to use the same name for different classes. This doesn’t happen when you are working on a small project because you declare all the classes with different name. It happens when you are working in a team or you imported any library to your project, if the developer uses the same class name in his library as you used in your project then same name class collision occurs. When we define a function or class without a namespace then it is considered as a global space.

**Syntax:**

namespace namespacename;

The **namespace** keyword followed by the **namespace** name is used to declare the namespace.

**Code:**

First create a file with name first.php and write the following code.

<?php

class A{

 function print(){

 echo "im at global space";

}

}

?>

Then create a file with name second.php and write the following code.

<?php

namespace second;

class A{

 function print(){

 echo "im at namespace";

}

}

?>

Then create a file with name object.php and write the following code.

<?php

 namespace second;

include 'first.php';

include 'second.php';

 $obj= new A;

 $obj->print();

?>

Output:

im at namespace

In the above example we created two files with name (first.php and second.php), in both the files we have declared the function with same name and use both the files in (object.php).But the general behavior in PHP produces an error that can’t declare the function again, but in file (second.php) we used namespace which creates a virtual directory, and by namespace we can use the function of file (second.php).

In this way when we import any library in our project then the same name function don’t collide with each other, as the namespace creates a virtual directory.